AOJS：JavaScriptのためのアスペクト指向プログラミング・フレームワーク

大橋 昭　久保 淳人　水町 友彦　江口 和樹　村上 真一　高橋 竜一
鷲崎 弘宜　深澤 良彰　鹿糠 秀行　小高 敏裕　永井 洋一
山本 里枝子　吉岡 信和　石川 冬樹　錠 尚史

JavaScript はスクリプト言語の一つであり，Web におけるクライアントサイドプログラミングに広く用いられている．一方で，プログラムの大規模化により，ロギングなどの横断的関心事を効率よく管理することが課題となっており，その解決手法としてアスペクト指向プログラミングが提案されている．JavaScript においてアスペクト指向プログラミングを実現するフレームワークは幾つか存在するが，元のプログラムからアスペクトに関する記述を完全に分離できないという問題がある．本稿では，JavaScript のためのアスペクト指向プログラミング・フレームワーク AOJS (Aspect-Oriented JavaScript) を提案する．AOJS は，従来のモジュールとアスペクトの完全分離記述を実現し，アスペクトの織り込み処理をプロキシ上で行うことにより，アスペクトの織り込みを強制する．AOJS を用いることで，既存のアプリケーションに修正を加えることなく動作ログを取得するなどの応用が考えられる．

JavaScript is one of major script languages, which is widely used for client-side programming on the Web. In programming, since cross-cutting concerns such as logging scatter on many modules of a program and tangle with other concerns, the maintainability of the program will decrease. Separation of cross-cutting concerns and core concerns is one of the challenge of programming. Aspect-oriented programming (AOP) has been proposed as a mechanism that enables the modular implementation of cross-cutting concerns. However, existing AOP frameworks cannot separate aspect's weaving designation from JavaScript program. In this paper, we propose an aspect-oriented programming framework for JavaScript achieving completely separated aspect description. AOJS guarantees the complete separation of aspects and other core modules by adapting proxy-based architecture for aspect weaving. AOJS realizes implementation of cross-cutting concerns such as logging without modifying programs of existing application.

AOJS:Aspect-Oriented Programming Framework for JavaScript

Akira Ohashi　Atsuto Kubo　Tomohiko Mizumachi　Kazuki Eguchi　Shinichi Murakami　Ryuichi Takahashi　Hironori Washizaki　Yoshiaki Fukazawa, 早稲田大学, Waseda University.
Hideyuki Kanuka, 株式会社日立製作所, Hitachi, Ltd..
Toshihiro Kodaka　Rieko Yamamoto, 株式会社富士通研究所, Fujitsu Laboratorics ltd..
Youichi Nagai, 日本電気株式会社 中央研究所, NEC Corporation.
Nobukazu Yoshioka　Fuyuki Ishikawa, 国立情報学研究所, National Institute of Infomatics.
Hisashi Ikari, 株式会社とめ研究所, TOME R&D Inc..

1　はじめに

JavaScript はスクリプト言語の一つであり，World Wide Web(WWW) におけるクライアントサイド・プログラミングに広く用いられている．例えば，Ajax(Asynchronous JavaScript and XML) [2] では，XmlHttpRequest を JavaScript によって操作することで，クライアント・サーバ間の非同期な通信が行わ

コンピュータソフトウェア, Vol.16, No.5 (1999), pp.78–83.
[ソフトウェア論文] 2009 年 12 月 25 日受付.

れリッチな Web アプリケーションが実現される．

プログラミングにおいて，複数のモジュールに同様の処理が出現することがある．複数のモジュールに出現する代表的な処理として，ログ出力，セッション処理等がある．これらの処理は，複数のモジュールに散らばり，かつ，各モジュールの本質的処理と混在するため，モジュールの独立性を低下させ，結果として各モジュールやプログラム全体の保守性を低下させる．このような，複数のモジュールに出現する処理を横断的関心事と呼ぶ．JavaScript によるクライアントサイド・プログラミングにおいても，プログラムの大規模化により，動作ログの取得などの横断的関心事を効率よく管理することが課題となっている．

アスペクト指向プログラミング (Aspect-oriented Programming; AOP) は横断的関心事の分離実装を可能にするプログラミング手法である [7]．アスペクト指向プログラミングでは，横断的関心事をそれぞれ独立にアスペクトとして分離記述することで，モジュールの独立性を高める．記述されたアスペクトは，記述から実行までのいずれかの段階で織り込み対象プログラムに合成 (weaving; 以下，織り込み) され，実行される．

WWW クライアントサイド・プログラミングにおいてアスペクト指向プログラミングを実現するフレームワークは幾つか存在する．JavaScript に関係するアスペクト指向プログラミングは，HTML 中のイベントハンドラとして JavaScript コードを織り込むもの [1] [5] [11] [18] と，JavaScript コードに対する JavaScript コードの織り込みを行うもの [6] [12] [13] [16] [17] [19] に大別できる．これらの枠組みのうち，前者は HTML を対象とし，JavaScript プログラムに対する織り込みができないため，本稿では JavaScript を対象とする後者の手法に注目する．

一般的に，JavaScript プログラムは，プログラマの手で記述され，Web サーバに配置され，Web サーバから JavaScript プログラムが送信され，送信された JavaScript プログラムを処理系 (例えばブラウザ) が解釈し実行する，という段階を経る．これらの手順には，JavaScript プログラムの記述から実行までにプログラムの変換を行う段階が存在せず，従って，コンパイル型言語のようなアスペクトの織り込みが不可能である．このことから，JavaScript プログラムに対する JavaScript コードの織り込みを実現する既存のアスペクト指向プログラミング・フレームワークでは，JavaScript の言語機能のみを用いてアスペクト織り込みを実現している．

しかし，これらの既存フレームワークは，アスペクト織り込みを実現する処理を 1 つの JavaScript プログラムにまとめて提供しているため，織り込み対象プログラムに対してこの JavaScript プログラムを読み込む記述が必要となり，結果として横断的関心事に関する記述を完全に分離できない．

本稿では，JavaScript のためのアスペクト指向プログラミング・フレームワーク AOJS(Aspect-Oriented JavaScript) を提案する．AOJS は，構文解析器をベースとした処理系でアスペクト織り込み処理を行い，対象となるプログラムを修正することなくアスペクト指向プログラミングを実現する．さらに，アスペクト織り込み処理系はプロキシ上に配備され，透過的にアスペクトの織り込み処理を行うことで，対象となる JavaScript プログラム全てにアスペクトの織り込みを強制する．

AOJS を用いることで，アスペクトと織り込み対象プログラムの完全分離記述が可能になり，織り込み対象となるプログラムを全く変更せずに処理の追加や変更を実現できる．JavaScript を用いる Web アプリケーションにおける Web ブラウザの動作ログ取得コードを，JavaScript プログラムを変更せずに織り込むといった利用が考えられる．また，プロキシ上で織り込み処理を行うことで，アスペクトが織り込まれた JavaScript プログラムが HTTP 通信上のみに存在することになり，直接編集を事実上不可能にし，アスペクトと JavaScript プログラムの分離記述の維持に貢献すると考えられる．

以降において，2 章で AOJS の概要について説明し，3 章で AOJS の性能に関する評価を行う．4 章では関連研究について述べ，5 章で本稿のまとめとする．

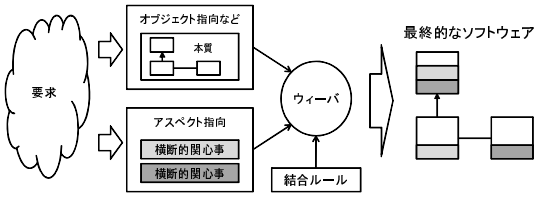

図 1　AOP の概要

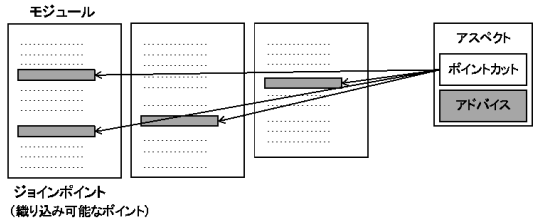

図 2　AOP の基本用語と関係

2　AOJS の概要

本章では，まずアスペクト指向プログラミングについて説明し，その後で提案フレームワーク AOJS の概要を説明する．

アスペクト指向プログラミング (以下 AOP) の概要を図 1 に示す．AOP とは対象を本質と横断的関心事に分離して捉え，結合ルールによって 1 つのプログラムへ合成するプログラミング手法である．複数のモジュールに横断して出現する処理を，新たなモジュール単位であるアスペクトとして記述し，プログラムの合成処理を行うウィーバによって後からアスペクトをモジュールへ織り込むことによって最終的なプログラム全体を得る．

AOP における基本用語とその関係を図 2 を用いて下記に示す．

アスペクト： ポイントカットとアドバイスの組み合わせを指定するモジュール

ジョインポイント： アドバイスの織り込みを指定可能な位置

ポイントカット： ジョインポイントの集合から実際の織り込み対象を絞り込む条件

アドバイス： ポイントカットによって選択されたジョインポイントへ織り込まれるコード断片

2.1　構成

AOJS におけるアスペクト織り込みは，Web サーバと Web ブラウザとの間に介在するプロキシサーバで行われる．

AOJS の概要を図 3 に示す．本稿では，Web サーバと同一の計算機で稼働するリバースプロキシとして AOJS を実現した．リバースプロキシとすることで，当該 Web サーバについて，JavaScript プログラムを含む全ての通信に介入できる．

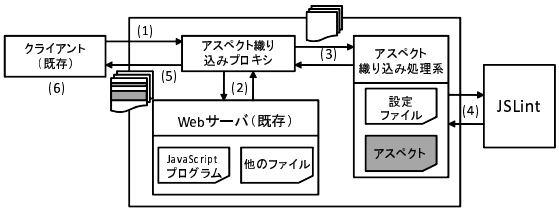

図 3　提案フレームワーク概要

Web ブラウザによる JavaScript プログラムの送信要求に対して，アスペクトが織り込まれた JavaScript が送信されるまでの順序を以下に示す．項番は図 3 の各数字に対応する．

1. **Web ブラウザからの JavaScript プログラム送信要求:** HTML ファイル等から参照されている JavaScript プログラムファイルについて，Web ブラウザはサーバに送信要求を行う．実際には，Web サーバの前段にあるアスペクト織り込みプロキシに対して送信要求がなされる．
2. **アスペクト織り込みプロキシによるファイルの代理取得:** アスペクト織り込みプロキシは Web ブラウザの代理として，Web サーバから当該ファイルの取得を行う．
3. **アスペクト織り込み:** 代理取得したファイルが JavaScript プログラムファイルだった場合，アスペクト織り込み処理系を起動してアスペクト織り込みを行う．開発者は，アスペクト織り込み処理系に対して，アドバイスとポイントカットを含むアスペクトを指定できる．
4. **アスペクト織り込み済 JavaScript プログラム**

の検証: アスペクトの織り込みによって書き換えられたファイルに構文エラーがないかを JSLint によって検証する．構文エラーがなければアスペクト織り込み済みプログラムが，構文エラーがあれば元の JavaScript プログラムが送信される．

5. **アスペクト織り込み済 JavaScript プログラムの送信:** アスペクト織り込み済 JavaScript プログラムを Web ブラウザに送信する．JavaScript プログラム以外のファイルは，変更無しに Web ブラウザに送信される．
6. **アスペクト織り込み済 JavaScript プログラムの実行:** Web ブラウザは受信した JavaScript プログラムを実行する．

AOJS は，プロキシにより Web サーバの通信に介入し，対象となる JavaScript プログラムを修正することなくアスペクトの織り込みが可能である．よって，すでに稼働しているアプリケーションに対しても容易にアスペクトを織り込める．このように AOJS は，単独で動作可能なプログラムに対して，ロギングのように後から付加できる関心事をアスペクトとして実装することを主に意図しており，したがって，元のアプリケーションをできるだけ止めないことを優先するため，項番の 4. に示すように，構文エラーの際には元の JavaScript プログラムを送信する．

対して，認証処理のように，対象となるアプリケーションの機能として必ず実装される必要がある関心事は，構文エラーにより最終的に出力されるプログラムに織り込まれない可能性があるため，AOJS による実装には適さない．

以降この章において，ポイントカット，アドバイス，JavaScript ファイルへのアスペクトの指定，織り込み対象の抽出，織り込み処理，織り込み処理されたプログラムの検証，キャッシュプロキシのための織り込み処理，AOJS の制限について述べる．

2.2 ポイントカット

横断的関心事として，クライアントの動作ログ送信を例にとる．動作ログにおいては，各変数の値の変化や関数の実行時間の測定が要求として挙げられる．また，サーバへログを送り返すための関数を新たに定義する必要がある．その他，契約による設計を例にとると，関数本体に対して事前条件や事後条件の挿入が必要となる．これらの処理を達成するためには，上記箇所を指定し追加コードの織り込みや処理の置換を実現できる必要がある．そこで AOJS について，ポイントカットとして JavaScript プログラム中の以下の要素を指定できるように設計した．

- 変数への値の代入
- 関数の呼び出し
- 関数の実行
- 処理対象 JavaScript ファイルの冒頭

図 4 は，AOJS における，アスペクトファイルの例である．アスペクトファイルは XML 形式で記述される．ポイントカットは aspectsetting 要素の子要素として記述され，要素名でポイントカットの種類を指定する．ポイントカット要素を記述できる個数に制限はない．さらに，各ポイントカット要素はその子要素としてアドバイスを持つ．1 つのポイントカット要素に対して記述できるアドバイス要素の個数には制限があり，詳しくは，具体的なアドバイス要素の記述方法を含め 2.3 節で述べる．以下に各ポイントカット要素についての説明を示す．

`var` **要素** 変数への値の代入を指定するポイントカット．varname 属性で変数名を指定し，子要素に後述のアドバイスを記述する．図 4 の 8 行目においては，y という変数への代入が行われる箇所がジョインポイントとして絞り込まれる．

`function` **要素** 関数呼び出し/実行を指定するポイントカット．functionname 属性で関数名を，pointcut 属性で関数の呼び出し元への織り込み (call) か関数本体への織り込み (execution) かを指定し，子要素にアドバイスを記述する．図 4 の 12 行目においては，fib_gen_2() という関数の呼び出しが行われる箇所がジョインポイントとして絞り込まれる．

`initializeFile` **要素** 処理対象 JavaScript ファイルの冒頭を指定するポイントカット．initializeFile 要素は子要素を持たず，対象 JavaScript プログラムの先頭に追加するアドバイスを記述したファイルの名前を要素内に記述する．図 4

```
<?xml version="1.0" ?>
<aspectsetting>
  <initializeFile>init.js</initializeFile>
  <var varname="/fib_gen_1/ret">
    <before><![CDATA[window.alert(__name__ + "@before: " + __beforeval__ + "<br />");]]></before>
    <after><![CDATA[window.alert(__name__ + "@after: " + __afterval__ + "<br />"):]]></after>
  </var>
  <var varname="/y">
    <before><![CDATA[document.write(__name__ + "@before: " + __beforeval__ + "<br />");]]></before>
    <after><![CDATA[document.write(__name__ + "@after: " + __afterval__ + "<br />");]]></after>
  </var>
  <function functionname="/fib_gen_2" pointcut="call">
    <before><![CDATA[var beg = (new Date()).getTime();]]></before>
    <after><![CDATA[var end = (new Date()).getTime();
                    sendLog( __retvalue__ + ", " + (end - beg) + "ms");]]></after>
  </function>
</aspectsetting>
```

図 4　アスペクトの例

の 3 行目において，init.js というファイルを対象 JavaScript プログラムの冒頭に追加することを指定している．なお，1 つのアスペクトファイルに複数の initializeFile 要素を記述できるため，アドバイスを複数のファイルに分割することも可能である．

AOJS では，1 つの JavaScript ソースファイルについて処理を行い，プログラムのインクルードや HTML の script 要素によって読み込まれる他の JavaScript プログラムについては考慮しない．1 つの JavaScript プログラムにおいて，プログラムの各要素は木構造をとる．最上位の要素として，グローバルな変数や関数等が定義できる．関数中にはさらに変数や関数を定義できる．従って，プログラム中の各要素は，ルート要素から各要素に至るパスによって特定可能である．ポイントカットはパスを用いて指定する．

AOJS におけるパス記述は，例えば`/myfunc/foo/x` のように行う．パス記述はスラッシュ記号から始まり，スラッシュ記号を区切りとして JavaScript プログラム中に出現する各要素について，上位から下位に向かって記述する．変数・関数名は直接記述の他，ワイルドカードと正規表現を用いて記述できる．ワイルドカードは`"*"`によって示され，変数名または関数名に使用できる文字から構成される任意の文字列にマッチする．スラッシュ記号は変数名に使用できずワイルドカードにはマッチしないため，ワイルドカードはスコープのネストを越えたマッチはしない．関数スコープをまたいでワイルドカードをマッチさせるには，`/*/*/x` のように，複数の`"*"`を記述する必要がある．正規表現は Java 言語の java.util.regex パッケージと同等の記述が可能である．パス記述においては，変数・関数を区別しない．パス記述は，var 要素や function 要素の varname 属性および functionname 属性で用いられる．

パス記述の例を以下に示す．

/y: グローバルな変数または関数 y

/fib_gen_1/*: グローバルスコープに存在する fib_gen_1 関数スコープ上で宣言されるすべての変数または関数

2.3　アドバイス

AOJS について，アドバイスとして before/after そして around の 3 つの要素を指定できるように設計した．例えば，関数の実行時間の測定においては，ジョインポイントの実行前後で時間を取得する必要があるため，ジョインポイントの直前と直後へのコード追加が必要となる．また，関数が持つ処理の内容を全く別の処理へ置換する要求もある．

アドバイスは，アスペクトファイル (図 4) におけるポイントカットの子要素として記述される．<before> のように，要素名でアドバイスの種類を指定し，要素中に織り込むコードを直接記述する．1 つのポイントカット要素に対しては，before と after アドバイスの両方を指定する場合のみ 2 つのアドバイス要素を記述可能で，それ以外の場合は，アドバイス要素を 1 つのみ記述する．同一箇所に複数のアドバイスを織り込む場合には，同じジョインポイントを指定するポイントカット要素を，複数記述する必要がある．

AOJS で指定できるアドバイスを以下に説明する．

- **before アドバイス：**
  ジョインポイントが実行されるより前の位置にコードが織り込まれ，実行時にはジョインポイントの実行直前に処理が行われる．図 4 の 5 行目では，var ポイントカット要素の下位に before 要素を指定しており，この例では，ret 変数への代入前に ret の値がアラートメッセージとして出力される．
- **after アドバイス：**
  ジョインポイントが実行される後の位置にコードが織り込まれ，実行時にはジョインポイントの実行直後に処理が行われる．図 4 の 6 行目では，var ポイントカット要素の下位に after 要素を指定しており，この例では，ret 変数への代入後に，ret の値がアラートメッセージとして出力される．
- **around アドバイス：**
  ジョインポイントの本来の処理は実行されず，要素内に記述した処理が代わりに実行されるようにコードが織り込まれ実行される．なお，around アドバイスは function ポイントカットのみで指定可能である．
  ジョインポイントが本来持つ処理は，__proceed__() 関数に格納され，around アドバイス内で呼び出して使用できる．図 5 の例では，foo() という関数の本体を，図に示す around 要素中のコードに置換するため，実行時に obj の値が null であったら，foo() 関数の処理は実行されない．

```
<function functionname="/foo"
    pointcut="execution">
  <around><![CDATA[
    if( obj !== null){
      __proceed__();
    }
  ]]></around>
</function>
```

図 5 around アドバイスの例

アドバイスにおいては，織り込み先のジョインポイントの情報が必要となる．そこで AOJS では，ジョインポイントの変数・関数名を__name__変数として，また，各ジョインポイントの実行に関わる値を，それぞれ以下に示す変数として参照できるように設計した．

- 変数代入ポイントカット (var)
  指定された変数の代入前の値を__beforeval__変数，代入後の値を__afterval__変数として参照できる．なお，__beforeval__変数は after アドバイス内でも参照可能である．
- 関数呼び出しポイントカット (call)
  関数の実行結果として得られる値を__retvalue__変数として参照できる．
- 関数実行ポイントカット (execution)
  関数実行前後の引数の値を参照することができ，対象となる関数の N 番目の引数はそれぞれ__beforeparamN__，__afterparamN__として参照できる．例えば，関数実行前の 1 番目の引数の値は，__beforeparam1__変数として参照できる．なお，引数がオブジェクトであった場合，__beforeparamN__は該当する引数と同じ参照を保持する．したがって，関数内の処理によってプロパティ等の値が変更される場合，after アドバイス内での__beforeparamN__の値も変更されていることに注意が必要である．

このように，特定の役割を持った変数をあらかじめ準備しておくことで，ワイルドカードによってポイントカットを指定する際，複数のジョインポイントが織り込み対象となった場合の変数参照にも対応できる．

また，これらの変数以外に，例えば，関数の実行時

間を測定するために，関数の呼び出し箇所の前後で時刻を測定する場合，時刻を格納しておく変数が必要となる．必要となる変数はアドバイス内で任意に定義できるが，元の JavaScript コードが持つ変数と名前が衝突するリスクを伴う．そこで AOJS について，アスペクト織り込み時にアドバイスごとに変数スコープを形成するよう設計しており，名前が衝突するリスクがないためアドバイス内で自由に変数を定義できる．スコープを形成する詳細については，2.6 節で述べる．

AOJS のアスペクト織り込み処理系は，アスペクトファイルを解析した後，before/after アドバイスから織り込み処理を行う．次いで around アドバイスが処理される．また，種類が同じアドバイスはアスペクトファイルで記述されている順に処理される．

AOJS は，プロキシ上ですべてのアスペクトを織り込み，プログラムの実行開始時にはすべてのアスペクトが有効となっているため，異なるジョインポイント間でのアドバイスの織り込み順序は出力されるプログラム全体に影響を与えない．しかし，同一のジョインポイントに対して複数のアドバイスを織り込む場合，アスペクトファイルにおける要素の記述順序とその種類は織り込み結果に影響を与える．

同一のジョインポイントに対して before/after アドバイスと around アドバイスの両方を適用する場合，around アドバイスは後から織り込み処理されるため，その織り込みの際には，before/after アドバイスがすでに織り込まれているジョインポイント全体を置換するようにコードが挿入される．したがって，実行時には around アドバイスのみが有効になる．ただし，__proceed__() 関数を呼び出すことで，before/after アドバイスを含むジョインポイントを呼び出すことができる．

同一のジョインポイントに対して同じ種類のアドバイスを複数適用する場合，アスペクトファイルにおいてより後に記述されたアスペクトは，すでに前のアスペクトが織り込まれているジョインポイントに対してアスペクトを織り込む．したがって，例えば，ジョインポイント JP に対して before アドバイスと after アドバイスを持つアスペクト A，B を A，B の順に記述して織り込んだ場合，最終的に実行されるコードは，beforeB，beforeA，JP，afterA，afterB の順に処理を行う．

2.4 JavaScript ファイルへのアスペクトの指定

JavaScript プログラムにどのアスペクトを適用するかは，図 6 の XML ファイルによって指定される．対象となる JavaScript ファイルと適用するアスペクトの対応関係は mapping 要素として記述され，mapping 要素の子要素に target 要素として JavaScript ファイル名を，aspect 要素としてアスペクトファイル名を記述する．当該プロキシサーバが扱うアスペクトと JavaScript の対応関係は，すべて configuration 要素の子要素に mapping 要素として記述し，この XML ファイルはプロキシサーバ上にただ 1 つだけ配備される．

アスペクト織り込み処理系は，入力として受け取った JavaScript のファイル名をこの XML ファイルから探し出し，一致するファイル名があれば，対応するアスペクトファイルを読み込み，記述されているポイントカットに基づいた織り込みを行う．なお，init.js などの initializeFile 要素によって指定される JavaScript ファイルを含むアスペクトファイルは，すべてプロキシサーバに配備される．

target 要素の JavaScript 名は，サーバのルートディレクトリからの相対パスで指定し，パス記述はポイントカットにおける記述と同等で，ワイルドカード・正規表現も使用できる．図 6 に示す`/Home/JS/[a-z]*.js`という例では，サーバのルートディレクトリにある Home ディレクトリ下の JS ディレクトリ下のファイル名がすべてアルファベットの小文字からなる JavaScript ファイルが，log.xml と対応づけられる．

2.5 織り込み対象の抽出

アスペクト織り込み処理系は，出現した変数や関数について，ポイントカットに適合するジョインポイントであるか否かを判定する必要がある．アスペクト織り込み処理系は，処理対象となる JavaScript プログラムとアスペクトを入力として受け取る．入力を受けた AOJS は，2 段階処理によってポイントカットに適合するジョインポイントを抽出する．

```
<?xml version="1.0" ?>
<configuration>
<mapping>
  <target>/fibonatti.js</target>
  <aspect>fibonatti.xml</aspect>
</mapping>
<mapping>
  <target>/Home/JS/[a-z]*.js</target>
  <aspect>log.xml</aspect>
</mapping>
</configuration>
```

図 6 JavaScript プログラムへアスペクトを指定する XML ファイル

第 1 段階では，JavaScript プログラムを先頭から順に解析し，それぞれの関数スコープ内で定義されている変数を抽出する．図 7 の JavaSciript プログラムを入力としたとき，図 8 の構造が得られる．図 8 における各ノードの上の区画はスコープを形成する関数名を，下の区画はそのスコープ内で宣言されている変数名を示している．入力 JavaScript プログラム全体は仮想ルートノードとして表現され，グローバル変数は仮想ルートスコープに含まれる．例えば，図 8 の中段の左端のノードは，仮想ルートスコープで宣言されている fib_gen_1 関数内で，変数 dummy と ret が抽出されたことを示しており，このノードは図 7 の 2〜7 行目に対応する．

第 2 段階では，JavaScript プログラム中での変数・関数の参照を抽出し，実際に参照・実行される関数・変数が，入力されたポイントカットに適合する場合にアスペクト織り込みを行う．例として，図 4 のアスペクトの各ポイントカットに適合するジョインポイントを表 1 に示す．

なお，アスペクトの織り込み処理によって追加されたコードが，自分自身や他のアスペクトのポイントカットの指定するジョインポイントを含む可能性がある．しかし，ジョインポイントの抽出は元の JavaScript プログラムに対してのみ行われ，追加されたコードをに対するジョインポイントの抽出は行われないため，織り込みによって追加されたコードへは，アスペクトは効果を発揮しない．

```
var x = 0; var y = 1;
function fib_gen_1() {
  var dummy, ret;
  ret = x + y; x = y;
  dummy = y = ret;
  return ret;
}
var z = 1;
function fib_gen_2() {
  function fib_2(x) {
    if(x <= 0) { return 1; }
    else if(x == 1) { return 1; }
    else {return fib_2(x - 1) + fib_2(x - 2);}
  }
  return fib_2(z++);
}
function fib_1() {
  var ret; ret = 100; y = 200;
  for(var i = 0; i < 30; ++i)
   document.form1.result.value = fib_gen_1();
}
function fib_2() {
  for(var i = 0; i < 30; ++i)
   document.form2.result.value = fib_gen_2();
}
```

図 7 フィボナッチ数計算を行う JavaScript プログラム

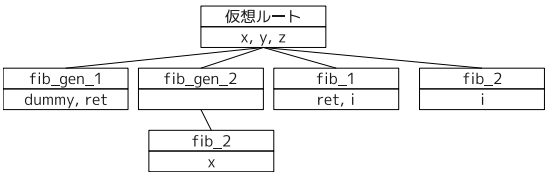

図 8 図 7 におけるスコープの包含関係と変数

例えば，関数 A に適用するアスペクト B において，関数 C を呼び出すコードを織り込むとする．この時，アスペクト D が関数 C の呼び出し (call) に対してアドバイスを織り込むとすると，元の JavaScript コードに含まれる関数 C の呼び出しに対してはアドバイスが織り込まれるが，アスペクト B によって織り込まれた関数 C の呼び出しへはアドバイスは織り込まれない．

表 1 各ポイントカットに適合するジョインポイント

ポイントカット	ジョインポイント	位置
<initializeFile>init.js</initializeFile>	-	図 7/先頭
<var varname="/fib_gen_1/ret">	ret = x + y	図 7/4 行目
<var varname= "/y">	y = ret	図 7/5 行目
<var varname= "/y">	y = 200	図 7/18 行目
<function functionname="/fib_gen_2" pointcut="call">	fib_gen_2()	図 7/24 行目

2.6 織り込み処理

抽出されたジョインポイントを置換するように織り込みを行う．織り込みにおいては，各ポイントカットに対応したコードテンプレートが用いられる．図 9 に関数呼び出しに対応するコードテンプレートを示す．図中の関数リテラルは，ジョインポイントを包み込み，関数呼び出し (<target>) の実行前後に before/after アドバイスをそれぞれ含む．ジョインポイントとなるコード断片が返り値を持つ場合は，__retvalue__変数に一旦保存して，アスペクトの実行終了後に返す．定義された無名関数は，テンプレート最後部の () によって即座に実行される．式の値を持つ点においては，代入式も関数呼び出し式も同等であるので，変数代入においても同様のコードテンプレートが用いられる．

織り込み処理で得られるコード断片は全体として関数呼び出し式になり，織り込み前の代入式/関数呼び出し式を置換コードの関数呼び出し式で置換するため，プログラムの他の部分への影響がない．従って，図 7/5 行目のような，多重代入においても織り込み処理が可能である．また，2.3 節で述べたように，アドバイスごとに形成される無名関数の変数スコープは，アドバイス内で自由に変数を定義することを可能にする．

関数実行ポイントカットでは，関数が定義されているブロックの最初と最後に before アドバイスと after アドバイスがそれぞれ織り込まれる．変数代入/関数呼び出しに対する織り込みとは異なり，関数リテラルは織り込まれずスコープが生じないため，アドバイス内で新たに変数を定義する場合は元のコードに干渉しないよう注意する必要がある．

例として，図 7 について，図 4 で指定されたアスペクトを織り込む．各ポイントカットで指定されるグローバル変数 (/y)，ローカル変数 (/fib_gen_1/ret)，関数呼び出し (/fib_gen_2) について，コードテンプレートを用いた織り込みが行われ，結果として図 10 の JavaScript プログラムが得られる．

```
(function(){
  var __name__ = "<target>";
  <before>
  var __retvalue__ = <target>;
  <after>
  return __retvalue__;
})();
```

図 9 関数呼び出し織り込み用コードテンプレート

2.7 織り込み処理されたプログラムの構文チェック

AOJS では，アドバイスとして任意のコードを元の JavaScript プログラムに織り込むことができる．アスペクト織り込み処理系から出力された JavaScript プログラムは，そのままクライアントへ送信され実行されるため，織り込み処理によって構文エラーが混入した場合，クライアントサイドで正確に実行されない可能性がある．そのためアスペクト織り込み処理系は，織り込み処理をしたプログラムを出力する前に，JSLint [10] を用いて構文のチェックを行う．

JSLint は JavaScript 用のコードチェッカーであり，構文エラーの他に未定義変数の使用などの好ましくないコーディングも検出し報告する．JSLint は ECMAScript の定義よりも厳密なサブセットを定義しているが，オプションによってユーザが許容したい JavaScript のサブセットの範囲を指定できる．このオプションの指定は検査対象の JavaScript ファイルの冒頭に記述される．AOJS では<initializeFile>ポイントカットを用いて，任意のファイルをプログラム

```
function sendLog(string) {
    var httpoj = createXMLHttpRequest();
    httpoj.open( "GET" , "/log.cgi?message=" + string, false );
    httpoj.send( "" );
}
(中略)
function fib_gen_1() {
  var dummy, ret;
  (function(){ var __name__ = "ret"; var __beforeval__ = ret;
               window.alert( __name__ + "@before: " + __beforeval__ + "<br />");
               var __afterval__=ret = x + y;
               window.alert( __name__ + "@after: " + __afterval__ + "<br />");
               return __afterval__;})();
  x = y;
  dummy = (function(){ var __name__ = "y"; var __beforeval__ = y;
               document.write( __name__ + "@before: " + __beforeval__ + "<br />");
               var __afterval__=y = ret;
               document.write( __name__ + "@after: " + __afterval__ + "<br />");
               return __afterval__;})();
  return ret;
}
(中略)
function fib_2() {
  for(var i = 0; i < 30; ++i) document.form2.result.value = (function() {
    var __name__ = "fib_gen_2";
    var beg = (new Date()).getTime();
    var __retvalue__=fib_gen_2();
    var end = (new Date()).getTime(); sendLog( __retvalue__ + ", "
    + (end - beg) + "ms");return __retvalue__; })();
}
```

図 10 織り込み処理された JavaScript プログラム (一部)

の冒頭に織り込めるため，オプション記述用のファイルを作成し後から織り込むことで，元の JavaScript ファイルには手を加えることなく JSLint の検証項目を変更できる．

図 11 に設定ファイルおよび指定できるオプションの例を示す．例えば，undef オプションを true にすることで，未定義変数の使用がチェックされる．オプションの一覧は [10] より参照できる．JSLint の構文チェックに成功すると，織り込み処理系はプログラムが品質上問題ないと判断し，そのままクライアントサイドへ送信する．構文チェックによりエラーが検出された場合は，織り込み処理前の元のプログラムを送信する．なお，構文エラーがあった場合には，JSLint によりエラー内容がプロキシサーバ上に出力されるため，開発者これを確認することでアスペクトの修正箇所を判断できる．

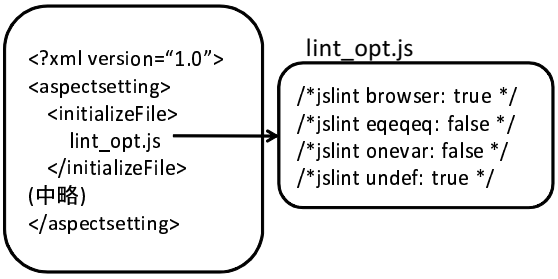

図 11 JSLint のオプションの例

2.8 キャッシュ処理のためのコード自動織り込み

AOJS は，プロキシ上でアスペクト織り込み処理を完了するが，アスペクト織り込み処理はジョインポイ

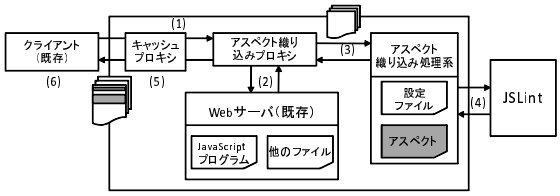

図 12　キャッシュプロキシ

ント抽出のための構文解析を含むため，特に大きなプログラムについて Web ブラウザへの応答時間を悪化させると予想される．そこで，AOJS ではキャッシュプロキシを用いることで応答時間の向上を図る．同一のリクエストに対しては，アスペクト織り込み処理の結果は変わらないため，最初のリクエストに対して計算した結果をキャッシュしておくことで，以降のリクエストに対してはキャッシュプロキシから即座にファイルを送信できる．キャッシュプロキシは図 12 のようにアスペクト織り込みプロキシの前段に配置する．

しかし，キャッシュプロキシを導入する際の問題として，アスペクトファイルを変更してもキャッシュ上のファイルには反映されず，古いファイルが残る可能性が挙げられる．AOJS は，プロキシ上でアスペクトを織り込むことで，アスペクトが織り込まれた JavaScript プログラムと元の JavaScript プログラムの整合性を保証するため，最新のアスペクトが織り込まれた JavaScript プログラムを，確実にクライアントに送信する必要がある．

この問題を解決するために，AOJS では，ファイルが変更されているかをチェックする CGI プログラムをサーバ上に配備する．この CGI プログラムは，ファイル名と時刻をパラメータに取り，指定されたファイルの最終更新時刻を取得し，パラメータで渡された時刻との比較を行う．そして，比較した時刻が同じであれば変更がないと判定し，時刻が異なれば変更があったと判定する．アスペクト織り込み処理系は織り込みの際に，この CGI プログラムをファイルの読み込み直後にリクエストするコードを，自動で対象となる JavaScript プログラムに追加する．この時，CGI プログラムのパラメータには，織り込み対象となるファイル名とアスペクトファイルの最終更新時刻が，織り込みの段階で割り振られる．この仕組みにより，すべてのキャッシュされるアスペクト織り込み済みプログラムには，アスペクトファイルが変更されているかの確認を行う機能が付加され，実行時には以下の順で処理が行われる．

1. クライアントは JavaScript プログラムを受信後，CGI プログラムのリクエストを行う．
2. サーバ上で CGI プログラムが実行される．対象となるアスペクトファイルの最終更新時刻を取得し，パラメータとして受け取った時刻との比較を行う．
3. 比較において値が異なる場合，アスペクトファイルが変更されたと判定し，キャッシュプロキシから該当するファイルのキャッシュを削除する．
4. サーバからクライアントへ判定結果を送信する．
5. クライアントは，サーバからのレスポンスを受け取り，変更がある場合には自動でページの再取得を行う．

再取得時には，すでに現在のページのキャッシュは削除されているため，キャッシュプロキシは AOJS プロキシへファイルを要求する．そして，アスペクト織り込み処理が行われ，新しいアスペクトが織り込まれたファイルがクライアントへ送信される．

2.9　制限

AOJS は，仕様として以下の項目に制限を持つ．

2.9.1　無名関数

無名関数を変数に代入することで関数を作成している場合，関数は名前を持たず階層に名前をつけることができないため，ポイントカットによる指定ができない．また，ジョインポイントとして抽出されないため，ワイルドカードですべての関数を指定しても，マッチングの対象にはならない．その他，無名関数を束縛する変数はジョインポイントとなるが変数として扱われるため，その呼び出しや関数の本体は call や execution ポイントカットの対象とはならない．

2.9.2　変数名の参照時修飾

配列やオブジェクトプロパティのように，変数宣言時と使用時で識別子が異なる場合，両者を同一の変数として判定できない問題がある．例えば，オブジェ

クトプロパティを参照する際には，プロパティ名の前に呼び出し元のオブジェクト名を記述しなければならない．しかし，AOJS は変数宣言時の識別子でジョインポイントの判定をするため，上記のように変数宣言時と変数使用時で識別子の記述が異なる場合は，同一の変数として判定されない．

ただし，代入式全体を含む関数を新たに定義し，その関数の呼び出しをジョインポイントとして指定することで，関数呼び出しに対する織り込みを経由しての代入式への織り込みが可能となり，この問題を解決できる．直接の織り込みは，今後のポイントカットの拡張で対象に含める予定である．

2.9.3 曖昧なコーディング

Web ブラウザに含まれる JavaScript 実行環境は，多くが曖昧な構文解析を行う．寛容な JavaScript パーサに依存して，多くのライブラリにおいて，行末セミコロンの省略など文法に沿わない部分があるため，提案システムで構文解析できない場合がある．前処理による文法に沿った形式への変換か，提案システムのパーサを変更するなどの対策が必要である．

2.9.4 インクルード順序の考慮

ライブラリを利用する JavaScript プログラムにおいては，JavaScript プログラムが Web ブラウザ上で統合される．したがって，複数の同名関数定義が存在した場合，実際に参照される関数がどの関数になるかは JavaScript ファイルのインクルード順序から影響を受ける．しかし，AOJS は JavaScript ファイル単体での織り込み処理を行うため，クライアント上での関数定義の上書きを考慮できず，AOJS によってアドバイスが織り込まれた関数が上書きされ，アドバイスが無効化される可能性がある．クライアントサイドでアスペクトを織り込む既存のフレームワークを組み合わせて用いることで，最終的に参照される関数に対して確実にアスペクトを織り込むことができるが，特定のライブラリを読み込むなどファイルの変更を必要とする．また，インクルード順序を実行時に解析する対策が考えられるが，サーバの応答時間を悪化させる上，当該ケースは頻繁に起こるものではないため，実装の必要性は低いと考えられる．

2.9.5 HTML 中に記述されたコード

JavaScript は，script 要素によって HTML ファイル中にも記述されるが，AOJS は JavaScript ファイルのみを解析するため，HTML ファイル中の JavaScript コードは織り込みの対象とならない．HTML 中から script 要素のみを抜き出して解析する手法が考えられるが，処理時間が増大すると予想されることや，サーバ側で該当部分を JavaScript ファイルとして記述しなおすことで対処できることから，AOJS では HTML 中の JavaScript コードを扱わない．

2.9.6 アスペクト間のデータの保持

AOJS では，ジョインポイントに直接アドバイスを追加することでアスペクト指向プログラミングを実現しており，アスペクトのオブジェクトやそのインスタンスは存在しない．したがって，例えば，ある関数が何回実行されたかをカウントする処理をアスペクトとして織り込む場合，合計回数を保持する変数はグローバル変数として確保することになる．

グローバル変数は，名前空間が干渉するリスクを伴うが，prototype.js [15] などのフレームワークに見られるように，グローバルオブジェクトを名前空間として用意し，そのプロパティとして必要な変数を定義することで干渉のリスクを回避できる．AOJS では，アドバイスの中で必要な変数およびそれを保持するオブジェクトを，initializeFile ポイントカットを用いることで容易に対象 JavaScript プログラムに追加できるため，開発者はこれに従うことが望まれる．

3 評価

本章では，AOJS の性能に関する評価を行う．Java 言語を用いてアスペクト織り込み処理系を，Perl 言語を用いて織り込みプロキシをそれぞれ実装した．JavaScript の解析処理系は，JavaCC を用いて生成したコード雛形を拡張して作成した．なお，想定する JavaScript のバージョンは 1.5(ECMA-262 3rd edition) である．

実験では，以下のマシンをローカルネットワーク内で 54Mbps のルータに接続して用いた．

サーバ: Intel Xeon 2.80GHz，2GB Memory，Debian/GNU Linux 5.0.5，Apache 2.2.9，Squid 2.7.

クライアント: Intel Core2Duo 2.4GHz，4GB Memory，MacOS 10.5.8.

3.1 アスペクトの完全分離記述

AOJS は，対象となる JavaScript プログラムに修正を加えることなくアスペクトの織り込みを実現する．本節では，再帰型のフィボナッチ数計算を行うプログラムに対して，計算時間を測定するコードをアスペクトとして分離し実装する例を示す．

使用するファイルは，図 7 の JavaScript プログラムと，図 4 のアスペクトであり，図 7 のプログラムは単独で実行可能である．これらのファイルと AOJS を Web サーバ上に配備し，Web ブラウザからプロキシサーバへアクセスする．JavaScript ファイルに対して図 4 のアスペクトが正確に織り込まれれば，Web ブラウザが受け取るプログラムは図 10 となり，fib_gen_2 関数呼び出しの前後で時刻の測定を行い，その差分をログとしてサーバへ送る処理が実行される．アスペクトが織り込まれてサーバへログが送信されるまでの流れを図 13 に示す．なお，sendLog 関数は XmlHttpRequest を用いてサーバにログを送信する関数であり，init.js ファイル中で定義される．

実行の結果，サーバに記録されたログを図 14 に示す．計算の進行に伴って計算時間が増大していく様子が出力されていることから，アスペクトとして分離した処理が Web サーバ上の JavaScript プログラムに織り込まれ，Web ブラウザ上で実行されたことが確認できる．アスペクト織り込みのために JavaScript プログラムの修正は行っておらず，AOJS により，アスペクトと JavaScript プログラムの完全分離記述が可能である．なお，対象となる JavaScript プログラムの修正が不要なため，既存の Web アプリケーションに対しても容易に適用可能である．

3.2 横断的関心事の実装・変更にかかるコスト

AOP を用いることで，横断的関心事を 1 つのモジュールにまとめて記述し管理できる．AOJS ではアスペクトは図 4 の形式のファイルに記述され，ワイルドカードを使用することによって，1 つのポイントカットで複数のジョインポイントを指定できる．本節では，アスペクトの記述にワイルドカードを用いてロギングコードを実装する例を示し，AOJS を用いることで，JavaScript ファイルに直接コードを記述する場合と比較して，より効率的な実装と変更が可能であることを確認する．

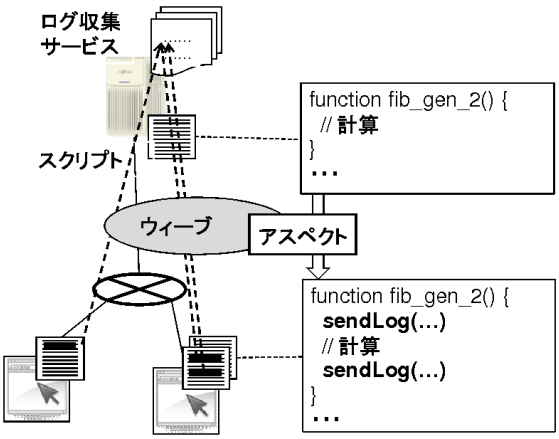

図 13 ウィーブからログ送信までの流れ

```
[Fri Jun 27 09:34:23 2008] [133.9.74.xx] 1, 0ms
[Fri Jun 27 09:34:23 2008] [133.9.74.xx] 2, 0ms
[Fri Jun 27 09:34:23 2008] [133.9.74.xx] 3, 0ms
[Fri Jun 27 09:34:23 2008] [133.9.74.xx] 5, 0ms
[Fri Jun 27 09:34:23 2008] [133.9.74.xx] 8, 0ms
(省略)
[Fri Jun 27 09:34:25 2008] [133.9.74.xx] 196418, 161ms
[Fri Jun 27 09:34:25 2008] [133.9.74.xx] 317811, 266ms
[Fri Jun 27 09:34:25 2008] [133.9.74.xx] 514229, 431ms
[Fri Jun 27 09:34:26 2008] [133.9.74.xx] 832040, 694ms
[Fri Jun 27 09:34:27 2008] [133.9.74.xx] 1346269, 1086ms
```

図 14 サーバに記録されたログ

本節では，ある企業の Web サイト†1に含まれる JavaScript プログラムに対して，図 15 の関数実行前に関数が実行されたことを出力するロギングコードを追加実装する．なお，得られた JavaScript ファイルは 5 つである．織り込むコードが複数のジョインポイントに対して共通である場合には，ワイルドカードを用いて複数のジョインポイントを同時に指定することで，重複するコード記述を避けられる．本評価実験では，get*，set*，init*となる関数をまとめて指定する．

ロギングコードを埋め込んだ JavaScript プログラ

†1 http://www.toshiba.co.jp/

```
<before><![CDATA[
    sendLog(__name__ + " -> execution");
]]></before>
```

図 15 関数実行を通知するロギングコード例

ムの実行の様子を図 16 に示す．本サンプルでは，アドバイス内で__name__変数を用いており，アスペクト織り込み時には各ジョインポイント固有の名前が代入されるため，どの関数が実行されたのかをログから判別できる．さらに，全てのロギングを図 17 の実行時間を測定するロギングコードに変更する．アスペクトとして実装したコードを変更する場合，AOJS ではアスペクトファイルに修正を加えるだけでよく，修正されたアスペクトは，Web ブラウザからのリクエストを受けてプロキシ上で再織り込みされる．

ここで，同様のロギングコードを，(1)JavaScript コードに直接記述する，(2)AOJS を用いて 1 つずつ織り込み先を指定する，という方法でそれぞれ実装し，上述の，(3)AOJS を用いてワイルドカードにより複数同時に織り込み先を指定する方法と，実装および変更における効率に関する比較を行った．比較内容は，実装時における JavaScript ファイルと XML ファイルへのコード記述量と他の関心事のコードを含まずに連続記述したコード行集合の数，および，変更時における変更箇所数と記述を変更したファイル数である．

比較結果を表 2 に示す．(2) と (3) について AOJS を用いる場合，本サンプルでは，ワイルドカードによって関数名を指定することで記述するコード行数が約 3 分の 1 になることがわかる．織り込むアドバイスが同一である場合には，ワイルドカードを用いることで効率的にコードを織り込むことができる．なお，コードの記述は XML ファイルに対してのみ行い，JavaScript ファイルへの記述を必要としない．これに対して，AOJS を用いない場合はロギングコードのみを記述するため，コードの記述量は最も少ない．しかし，JavaScript ファイルへの直接の記述を必要とする．

次にロギングに変更を行う場合のコストを比較する．

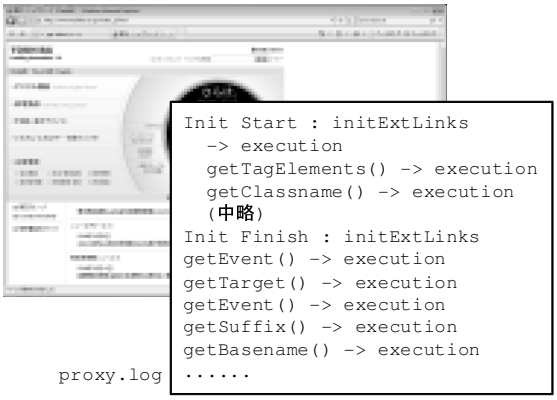

図 16 ロギングが実行される様子

```
<before><![CDATA[
  var beg = (new Data()).getTime();
]]></before>
<after><![CDATA[
  var end = (new Data()).getTime();
  sendLog(__name__ + "executed"
    + (end-beg) + "ms");
]]></after>
```

図 17 実行時間を測定するロギングコード例

(3) の AOJS でワイルドカードを用いるケースでは，他の場合と比べて約 4 分の 1 のコストで変更が完了できる．また，(1) と (2) では変更が必要な箇所数は同じであるが，(2) の場合は XML ファイルのみを変更すればよいため，JavaScript プログラム中から変更箇所を探し出すコストがかからない．その他，AOJS を用いることにより，織り込み対象の JavaScript ファイルそのものには修正を必要としない．

最後に，ロギングコードの散らばりを比較する．AOJS では，横断的関心事を実現する処理は，アスペクトとして 1 つのファイルにまとめて記述できる．この他には設定ファイルへ JavaScript ファイルへの対応付けを記述すればよいため，ロギングに関して記述したコード行集合は 2 つとなる．ロギングの変更を行う際は，修正を加えるのはアスペクトファイル 1 つだけで済むため，変更ファイル数は 1 となる．これに対して AOJS を用いずに直接 JavaScript ファイ

表 2　横断的関心事の実装および変更にかかるコスト

	実装			変更		
	記述量 [行]		コード行集合の数	変更箇所 [箇所]		変更ファイル数
	JavaScript	XML		JavaScript	XML	
(1) AOJS 未使用	22	-	22	22	-	5
(2) ポイントカット直接指定	0	75	2	0	22	1
(3) ポイントカット複数指定	0	26	2	0	5	1

表 3　織り込み処理時間の比較

対象	行数	所要時間 [秒]
図 7 のプログラム	25	0.364
rollover.js	125	0.736
uncss.js	29	0.309
new_win.js	52	0.399
topmain.js	30	0.428
bmv12.js	127	0.673

ルへロギングを記述すると，コード行集合は 22 個となり，プログラム中の複数箇所へ記述が必要なことが確認される．また，変更が必要となるファイル数は 5 つであり，すべてのロギングを実装・変更するためのコストが大きくかかる．

3.3　織り込み処理効率

実験用 Web サーバ上で，アスペクト織り込み処理の所要時間を測定し，AOJS がサーバのリクエスト処理効率に与える影響を評価する．対象となる JavaScript プログラムは，3.1 節で用いた図 7 の JavaScript プログラム，および，3.2 節で用いた 5 つのプログラムで，それぞれ図 4，図 15 のアスペクトを織り込む際の所要時間を測定する．測定は，各プログラムにつき 20 回ずつ行い，その平均値を算出した．

結果を表 3 に示す．2 列目は対象となるプログラムの行数を，3 列目はアスペクト織り込みの所要時間を示す．結果から，各アスペクト織り込み処理には数百ミリ秒かかることが確認される．AOJS は，ジョインポイントの抽出のために対象プログラムの構文解析を行うため，より大きなプログラムほどアスペクトの織り込み処理に時間を要し，表 3 からは，他と比べて行数の多い rollover.js と bmv12.js の所要時間が大きいことから，そのことが確認される．

この結果は，サーバにおけるリクエスト処理に遅延を生じさせることを示しており，特に 1 つの Web ページが複数の JavaScript を含んでいる場合には，遅延時間は各織り込み処理時間の合計となるためより大きな遅延となる．例えば，3.2 節で用いた Web サイトについて，含まれる 5 つのプログラムに対する織り込み処理時間はそれぞれ数百ミリ秒であるが，その合計は約 2.5 秒となり，アスペクト織り込み処理によってサーバのリクエスト処理効率は大きく低下する．

3.4　リクエスト処理効率

アスペクト織り込み処理によるサーバのリクエスト処理効率の低下を防ぐために，AOJS は 2.8 節に示すようにキャッシュプロキシによるパフォーマンスの向上を行う．本節では，キャッシュプロキシによるリクエスト処理効率の変化を確認するため，実験用 Web サーバ上に存在する 3.1 節と 3.2 節で用いた Web ページに，(1) 直接アクセスした場合，(2) 同一コンピュータで稼働する AOJS 経由でアクセスした場合，(3) さらにキャッシュプロキシを追加してアクセスした場合について実験を行った．それぞれの場合について，クライアントサイドのマシンからブラウザで Web ページを閲覧し，リクエストを出してからページ全体を読み込むまでにかかった時間を測定した．

測定では，アスペクトファイルの変更によるサーバの応答時間の変化を確認するため，5 分毎にアスペクトファイルを変更し，Web ページへのアクセスは 1 分毎に行う．実験時間は 100 分間とした．なお，アスペクトを変更する時刻にアクセスは行わない．測定時間は，アスペクトファイルを更新した時刻 T を基準として，ページを閲覧した時刻を T-2，T-1，T，T+1，T+2 分で分類をし，各測定値の平均値を算出した．

表 4 応答時間の測定結果

	ページ閲覧時刻	T-2	T-1	変更時刻 T	T+1	T+2
3.1 節の Web ページの読み込み時間 [秒]	Web サーバ	0.071	0.075	-	0.074	0.073
	＋ AOJS プロキシ	0.600	0.606	-	0.608	0.614
	＋キャッシュプロキシ	0.169	0.172	-	0.849	0.171
3.2 節の Web ページの読み込み時間 [秒]	Web サーバ	0.980	0.985	-	0.992	0.961
	＋ AOJS プロキシ	4.353	4.412	-	4.459	4.404
	＋キャッシュプロキシ	1.405	1.317	-	6.547	1.293

表 4 に測定結果を示す．単純に AOJS プロキシを追加した場合，大幅な性能低下が観察された．Web サーバに直接アクセスした場合と比較して，表 3 のアスペクト織り込み処理時間以上の差があるのは，Web サーバへの代理接続と JavaScript ファイルかどうか判定による追加の遅延が生じるからである．なお，アスペクトの織り込み処理はすべてのリクエストに対して行われているため，アスペクトファイルの変更による応答時間への影響はない．

squid によるキャッシュプロキシを追加した場合，アスペクトファイルを変更した直後を除くリクエストに対して，Web サーバに直接アクセスする場合に近い性能を得た．差分の遅延は，アスペクトの変更があるかの確認を行うための通信によるもので，本実験では 1 回の通信に 0.1 秒弱要した．複数の javaScript を含むページでは，その数だけ通信が行われるため，3.2 節の Web ページに対しては，ページ全体で 0.3〜0.4 秒の遅延が発生した．

一方，アスペクトファイルが変更された直後のアクセスのみ，単純に AOJS プロキシを追加した時と同様に性能低下が観察された．これは新しいアスペクトの織り込み処理が行われていることを示しており，キャッシュプロキシを用いた場合でも常に最新のアスペクトが織り込まれたプログラムを取得できることが確認された．

以上より，キャッシュプロキシを用いることで，織り込み処理は初回アクセス時の 1 回だけとなり，AOJS プロキシのみの場合と比較して性能の向上を実現した．

4 関連研究

ajaxpect [6]，ulibjs Aspect [12]，Cerny.js [16] および AspectJS [19] は，ポイントカットとしてオブジェクト・メソッドを指定可能なアスペクト指向プログラミング・フレームワークである．アスペクトの指定に先立って特定のライブラリを取り込んでおき，アスペクト全体を JavaScript コードとして記述する．

AspectScript [17] は，上記フレームワークと同じく JavaScript の関数としてアスペクトを記述するが，AOJS 同様，変数代入をポイントカットとして扱える．なお，両者はその機能を実現した最初期のフレームワークである．また，JavaScript の特徴である第一級関数をアスペクトの記述に使用できたり，ユーザが指定したイベントをジョインポイントとして定義できるなど，より細かなアスペクトの表現が可能である．

Dojo [1]，Ext JS [5]，The Yahoo! User Interface Library (YUI) [18] は，id 属性や name 属性など，特定の条件を満たす HTML タグのイベントハンドラをジョインポイントと見なして JavaScript コードを埋め込む．岡本らのアスペクト指向プログラミングフレームワーク [13] では，上記のイベントハンドラ埋め込みの条件指定の制約を緩和し，任意の HTML タグのイベントハンドラをジョインポイントとして指定可能である．

Mozilla Firefox は，JavaScript の独自拡張として watch/unwatch 関数 [11] を提供している．これらの関数を用いることで，オブジェクトのプロパティについて値の変化時に呼び出されるフックを設定および解除できる．対して AOJS は，オブジェクトのプロパティでない通常の変数について代入時を指定してコードを織り込める．また，AOJS は Web ブラウザの独自拡張機能を使わないため，より汎用的である．

上記の関連研究では，織り込み対象のプログラムや HTML ファイルに対して何らかの追加記述が必要である．例えば，当該アスペクト指向プログラミングフレームワークを実現する外部 JavaScript プログラムを読み込む記述が必要とされる．これに対して AOJS

は，織り込みに際して既存プログラムや HTML ファイルの変更が不要である．反面，AOJS では通信に介在するプロキシ上でアスペクト織り込みを処理しているため，JavaScript の言語機能のみを用いたアスペクト指向プログラミングと比較して，サーバの応答時間が大幅に低下する．ただし，3.4 節で示すように，キャッシュプロキシを導入することで，初回アクセス以降は通常の JavaScript ファイルをリクエストした場合と同等の応答時間を得られる．また，AOJS はプロキシ上で全てのアスペクト織り込み処理を完了するため，関連研究に見られるクライアント上でのアスペクト織り込みを必要としない．

大村らは，Web ブラウザと Web サーバの通信を関数呼び出しと見なすアスペクト指向ウェブフレームワークを提案した [14]．大村らのフレームワークと AOJS は，プロキシの形態で実現されている点で共通しているが，織り込み処理対象が HTML と JavaScript プログラムである点で異なっている．

Stamey [9] らは，PHP を用いて HTML 中に JavaScript コードを織り込む手法 Aspect-oriented PHP(AOPHP) を提案した．対象言語が PHP である点で AOJS と異なっているが，サーバからの送出時に織り込み処理を行う点で共通している．

AjaxScope [4] は，JavaScript コードについてオンザフライ式にプロキシによって AST 上でコード埋め込みを可能とする仕組みである．AjaxScope では AST 上の任意のノードを書き換え可能であり，例えばパフォーマンスプロファイリング処理のような新たなコードを追加導入できる．AjaxScope の基本的な仕組みは AOJS と同様であるが，AOP の枠組みではないため，プログラマの関心事に対応した新たなアスペクトを定義することはできない．

外村らは，Web アプリケーションに固有のポイントカットをまとめた枠組みである AOWP の PHP 言語用の実装として，AOWP/PHP を提案した [8]．Web アプリケーション向けの AOP フレームワークである点で AOJS と共通しているが，織り込み対象言語が PHP である点で異なる．また，実行前に織り込みを行う点が，プロキシの形態をとる AOJS とは異なる．

5 おわりに

本稿では，アスペクトを完全に分離記述可能な JavaScript のためのアスペクト指向プログラミング・フレームワーク AOJS を提案した．AOJS はリバースプロキシとして動作し，Web サーバからの JavaScript コード送出に割り込んでアスペクト織り込みを行うため，既存コードの変更が一切不要な上，対象となる JavaScript プログラムに対するアスペクトの織り込みを強制する，さらに，織り込み処理後の JavaScript プログラムが HTTP 通信上にしか存在しないため，織り込み処理結果の直接編集が事実上不可能になり，アスペクトと JavaScript プログラムの分離記述の維持に貢献できると考えられる．

3 章の評価実験から，計算時間の測定や複数箇所に散らばるログの記録など，要求に合致した織り込みができることを確認した．プロキシによる織り込み処理は性能を低下させるが，前段にキャッシュプロキシを設置することで，Web サーバに直接アクセスした場合に近い性能が得られることを確認した．AOJS を用いることで，JavaScript におけるアスペクト指向プログラミングの導入および既存アプリケーションへの容易なアスペクトの適用が期待される．

今後の課題として，以下が挙げられる．

- 無名関数のパス指定：無名変数への参照を保持する変数を経由して呼び出す必要があるため，参照を保持する変数の名前を階層名として利用できるが，より詳細な解析が必要である．
- 変数名の参照時修飾への対応：より詳細な構文解析や，各変数が同一かどうかの判定処理が必要である．
- 文法に厳密に沿わないコーディングへの対策：前処理によりソースコードを変更する方法と，提案システムの構文解析部を寛容にする方法がある．
- AOJS の応用：江口らは，Ajax パターンのモジュール化と適用において AOJS を部分的に使用している [3]．今後このような応用事例を通じて，他の様々な用途を検討していく予定である．

謝辞 本研究の一部は，財団法人 情報科学国際交流財団・産学戦略的研究フォーラム (SSR)2008，2009年度の助成を受けました．本研究の一部は国立情報学研究所・GRACE センターの支援を受けました．また，論文の改善に向けて，査読者の方々より有益なご助言・コメントを頂きました．

参 考 文 献

[1] Dojo : http://dojotoolkit.org/
[2] Edmond Woychowsky : AJAX: Creating Web Pages with Asynchronous JavaScript and XML, Prentice Hall, 2006.
[3] 江口和樹, 久保淳人, 鷲崎弘宜, 深澤良彰 : アスペクト指向による Ajax デザインパターンの適用, 第 16 回ソフトウェア工学の基礎ワークショップ FOSE 2009, 2009.
[4] Emre Kiciman and Benjamin Livshits : AjaxScope: a platform for remotely monitoring the client-side behavior of web 2.0 applications, *Proc. 21st ACM SIGOPS Symposium on Operating Systems Principles*, 2007, pp.17-30.
[5] Ext js : http://extjs.com/
[6] Google ajaxpect : http://code.google.com/p/ajaxpect/
[7] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. : Aspect-oriented programming, In *Proceedings European Conference on Object-Oriented Programming,* Springer-Verlag, Vol.1241(1997), pp.220-242.
[8] 外村慶二, 成瀬龍人, 塩塚大, 白石卓也, 鵜林尚靖, 中島震 : Web アプリケーション開発向け AOP 機構の実装, 情報処理学会第 163 回ソフトウェア工学研究発表会, SIG-SE-163-9, 2009.
[9] John Stamey, Bryan Saunders, and Simon Blanchard : The aspect-oriented web, In *the 23rd annual international conference on Design of communication: documenting designing for pervasive information*, 2005, pp.89-95.
[10] JSLint : http://www.jslint.com/
[11] Mozilla Foundation, Core javascript 1.5 reference:global objects:object:watch : http://developer.mozilla.org/ja/docs/Core_JavaScript_1.5_Reference:Global_Objects:Object:watch
[12] 岡田耕一 : ulibjs aspect : http://www.sip.eee.yamaguchi-u.ac.jp/kou/JavaScript/Aspect.html
[13] 岡本隆史, 鷲崎弘宜, 深澤良彰 : JavaScript におけるアスペクト指向プログラミングの応用, ウィンターワークショップ 2008・イン・道後 論文集, 情報処理学会ソフトウェア工学研究会, 2008, pp.69-70.
[14] 大村裕, 山岡順一 : アスペクト指向ウェブフレームワークによる情報共有システムの実現. ソフトウェア工学の基礎 XII, 2005, pp.13-18.
[15] Prototype Core Team : prototype.js : http://prototypejs.org/
[16] Robert Cerny : Cerny.js : http://www.cerny-online.com/cerny.js
[17] Rodolfo Toledo, Paul Leger, and Eric Tanter : AspectScript: Expressive Aspects for the Web, In *the Proceedings of the 9th ACM International Conference on Aspect-Oriented Software Development (AOSD 2010)*.
[18] The yahoo! user interface library : http://developer.yahoo.com/yui/
[19] Zero. Aspectjs : http://zer0.free.fr/aspectjs/